Операционные системы

лекция 5

hexlet.org

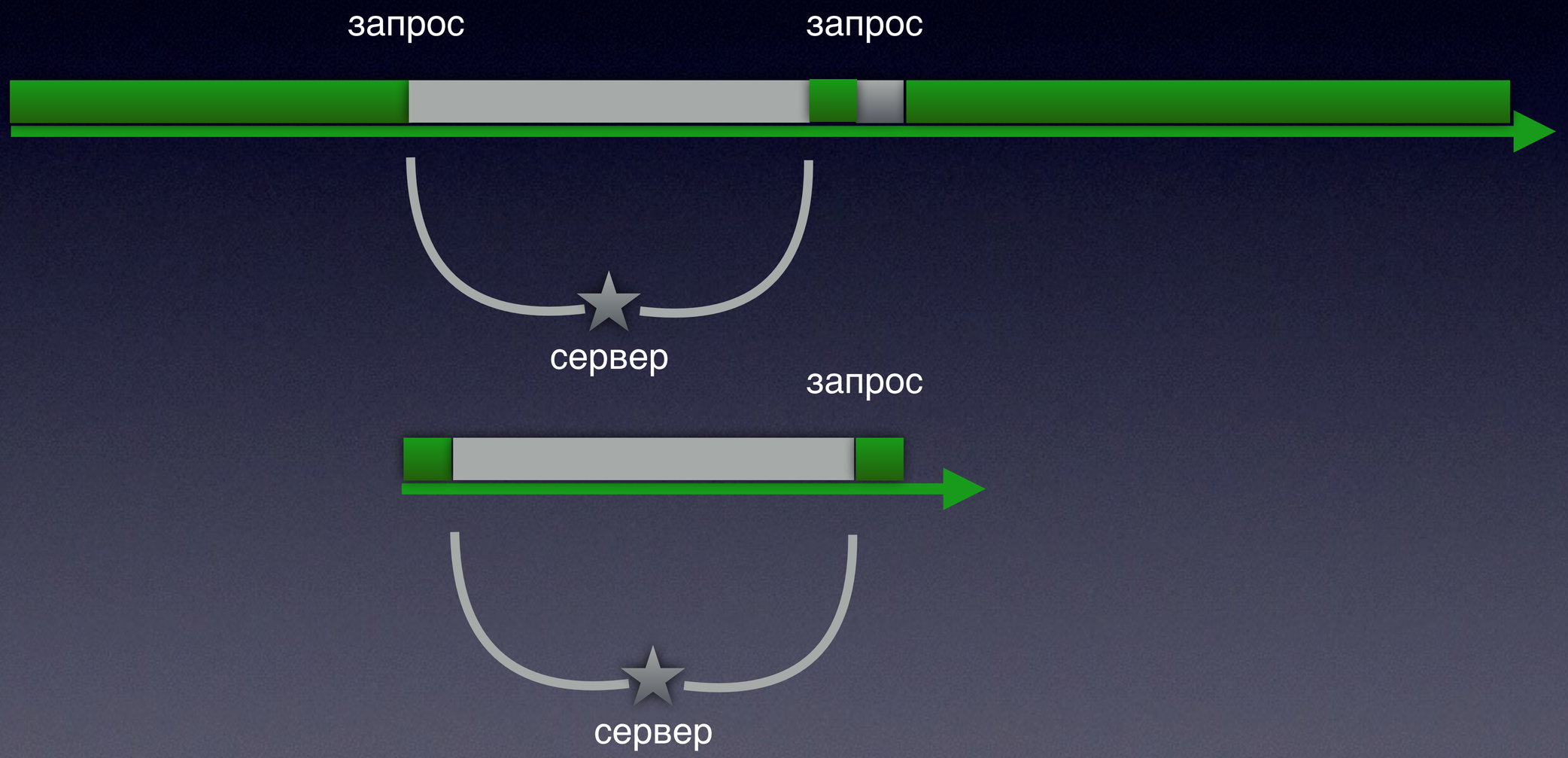
- Проблема решается с помощью замка
- Если закрыто – жди

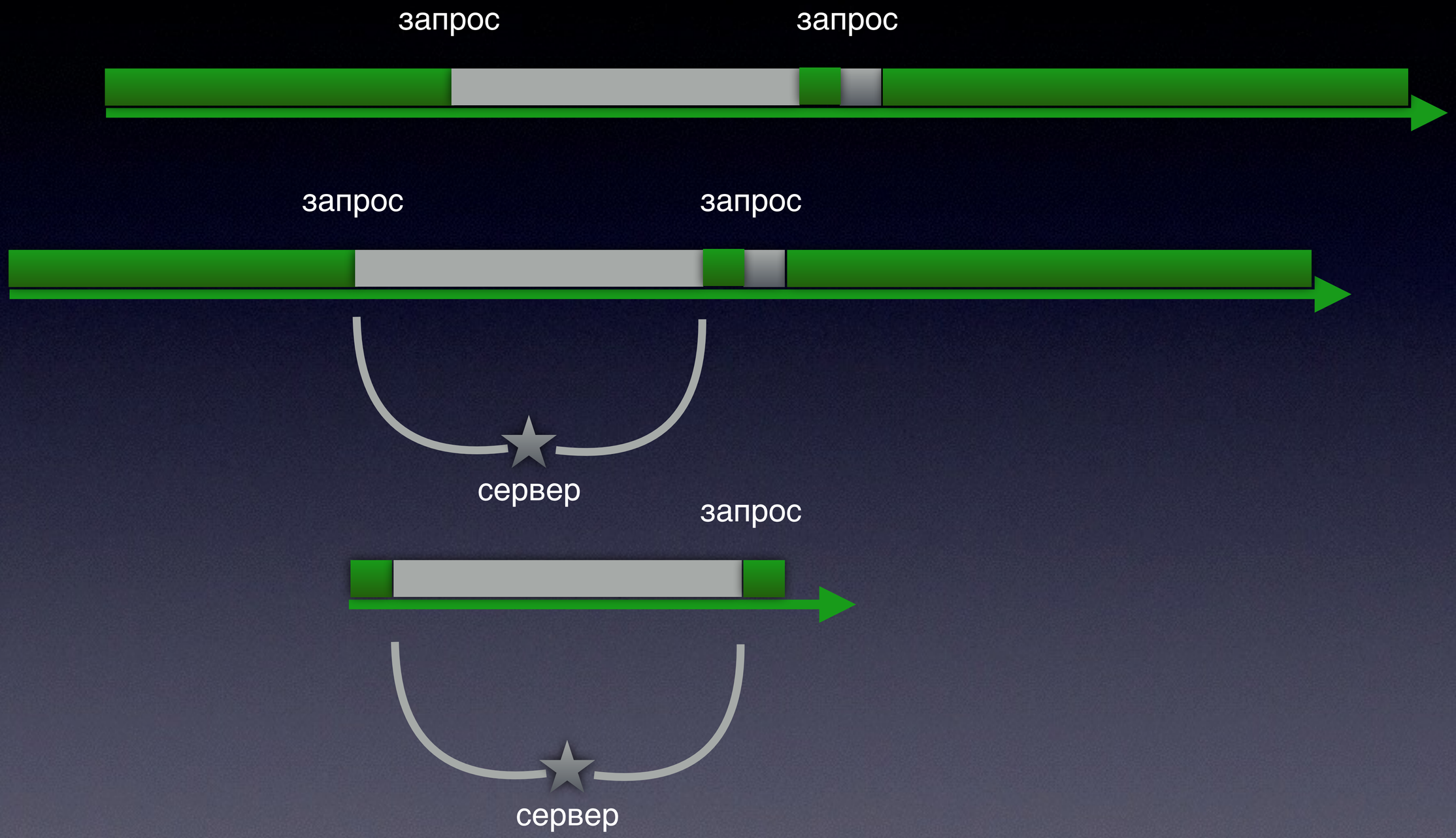
В целом свойства системы таковы

- Несколько агентов
- Общие ресурсы
- Правила доступа к ним

Несколько процессов

- Центральная проблема (задача)* современных операционных систем
  - Multiprogramming
  - Multiprocessing
  - Распределенные вычисления
- Параллелизм!
  - Управление взаимодействием нескольких процессов


запрос
запрос
сервер
запрос
сервер


запрос
запрос
запрос
запрос
сервер
запрос
сервер

Терминология

- Атомарная операция (atomic operation)
- Критическая секция (critical section)
- Взаимная блокировка (deadlock)
- Livelock
- Взаимное исключение (mutual exclusion)
- Состояние гонки (race condition)
- Resource Starvation

Проблемы

- Распределение глобальных ресурсов
- Сложно обнаружить ошибки в программе так как результаты не всегда детерминистические и воспроизводимые

Простой пример

- Две программы выполняются на параллельных процессорах, обе программы выводят что-то на экран
- Возможное решение: разрешить доступ только одному процессу в один момент времени

Заботы ОС

- Следить за процессами
- Выдавать и освобождать ресурсы
- Защищать ресурсы и данные от других процессов
- Сделать так, чтобы результаты выполнения не зависели от скорости выполнения

Конкуренция за ресурсы

Три главные проблемы:

- Необходимо взаимное исключение
  - критические секции
- Взаимная блокировка (Deadlock)
- Starvation

Что нужно для взаимного исключения

- Только один процесс может находиться в критической секции для ресурса
- Процесс, который завершается в некритической секции не должен мешать другим процессам
- Нет взаимной блокировки и ресурсного голодания
- Процесс не должен ждать доступа к критической секции ресурса если он свободен
- Не должно быть никаких допущений о последовательности и количестве процессов
- Процесс может находиться в критической секции ограниченное количество времени

Блокировка (Lock)

- Два состояния:
  - занято
  - свободно
- Две операции:
  - занять (перевести в состояния “занято” или ждать освобождения
  - освободить (перевести в состояние “свободно”)

Использование блокировок

- занять
- выполнить критическую секцию
- освободить

Использование блокировок

- блокировка защищает данные
- процессы (потоки) могут занимать разные блокировки
- нужно использовать одну блокировку для одного всех критических секций используемых одни и те же ресурсы

Пример: счет в банке

снять сумму со счета

- новый баланс = текущий баланс
- новый баланс = новый баланс - сумма
- текущий баланс = новый баланс

критическая секция

Пример: счет в банке

снять сумму со счета

acquire lock

➡ новый баланс = текущий баланс

➡ новый баланс = новый баланс - сумма

➡ текущий баланс = новый баланс

критическая секция

release lock

Что нам нужно в критической секции?

- Взаимное исключение
- Прогресс *(процесс вне критической секции не может запретить другому процессу войти в критическую секцию)*
- Конечное ожидание
- Производительность
- “Справедливость”

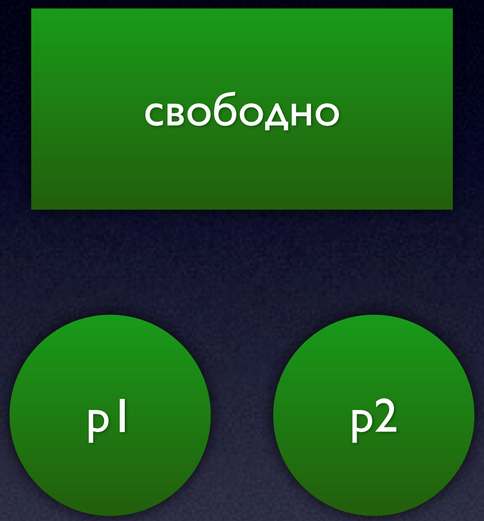
Пример реализации блокировки

```
struct lock {
  bool held;
}

void acquire(lock) {
  while (lock->held) wait;
  lock->held = true;
}

void release(lock) {
  lock->held = false;
}
```

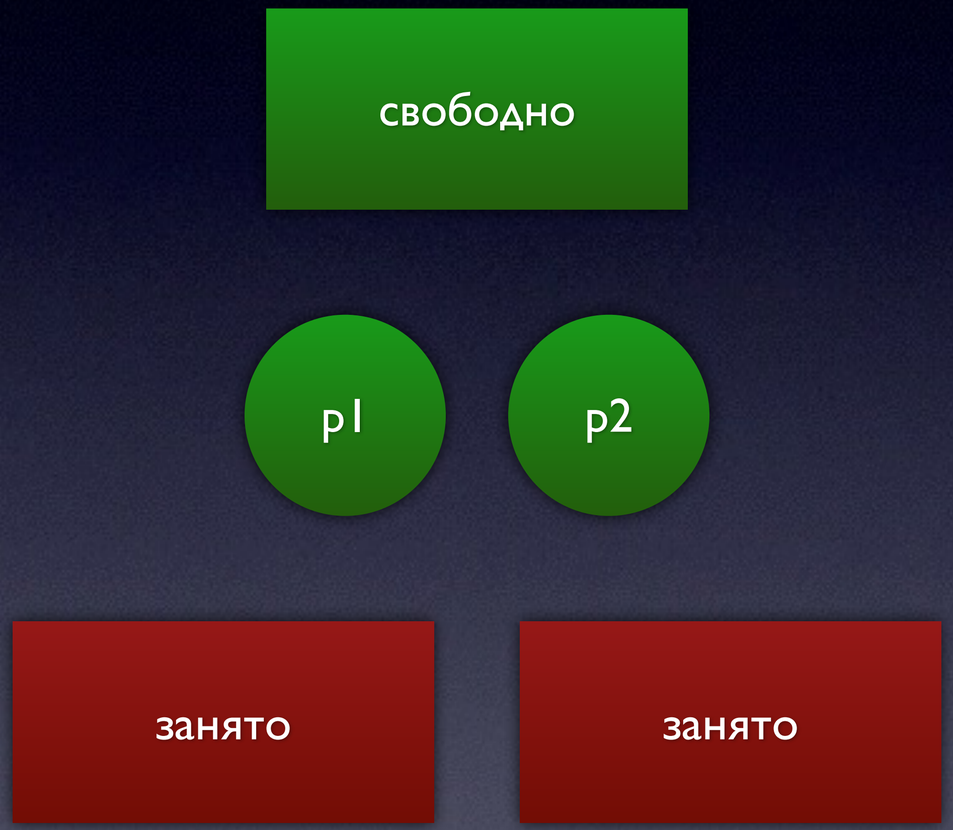
Проблема

свободно

Проблема



Проблема



Test-and-set

```
TestAndSet(flag) {

    oldFlag = flag;
    flag = LOCKED;

    return oldFlag;
}
```

Семафор

- Целочисленный счетчик, ограничивающий количество процессов, которые могут войти в определенный участок кода
- Только три (атомарные) операции:
  - инициализация
  - увеличение (signal)
  - уменьшение (wait)
    - если 0 – блокировка

Семафор

```
semaphore.init(5);


void DoSomething( void )
{
   semaphore.enter();
   ...
   semaphore.leave();
}
```

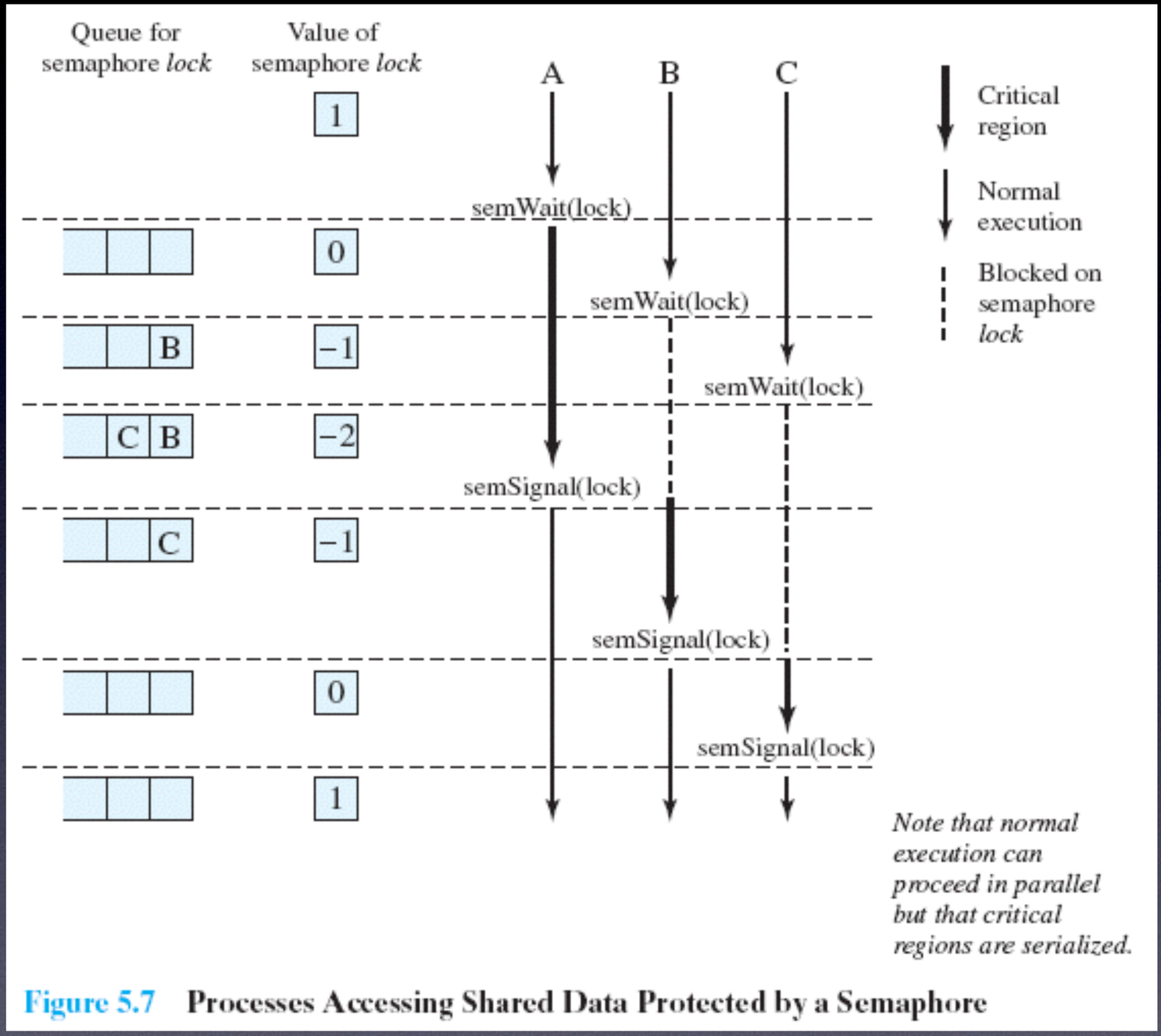
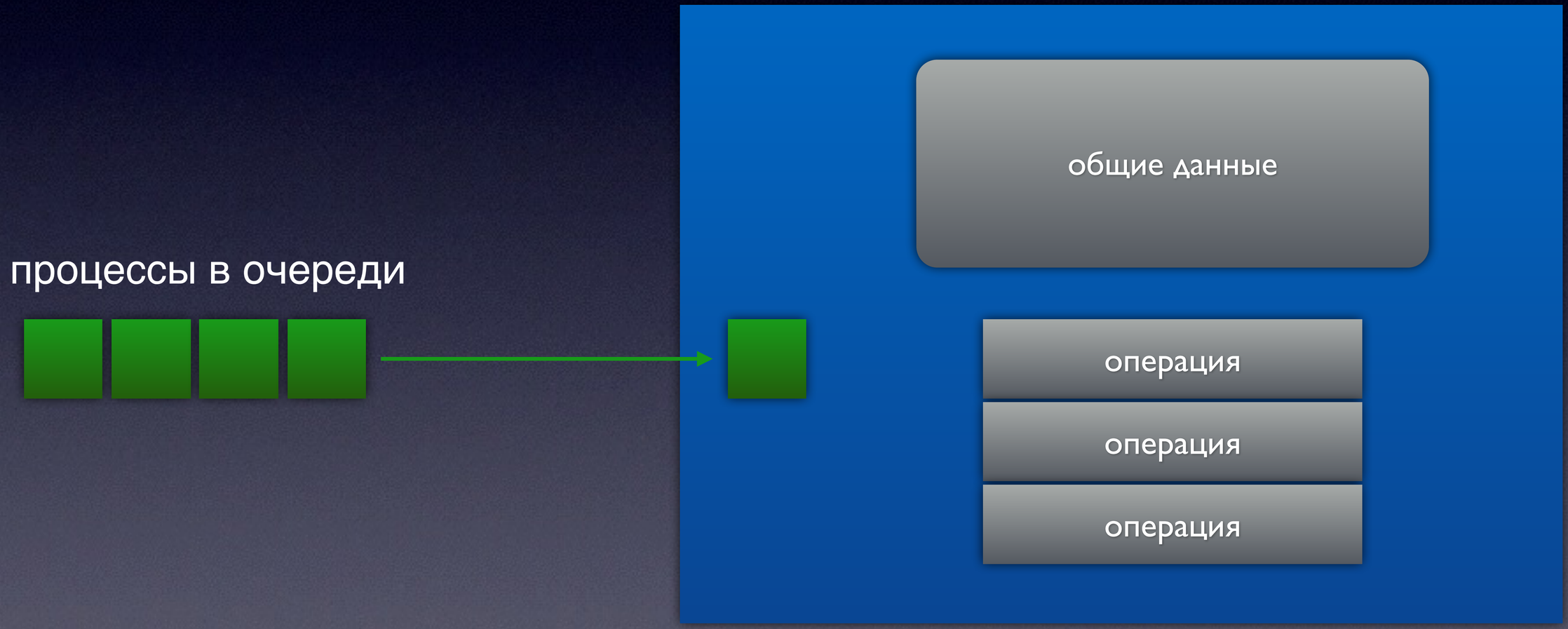


Figure 5.7 **Processes Accessing Shared Data Protected by a Semaphore**

Иногда этого недостаточно

- Процесс может ожидать какого-то события (удовлетворения какого-то условия)
- Семафоры усложняют программирование
- Иногда приходится использовать несколько семафоров в одной задаче

Монитор


процессы в очереди
общие данные
операция
операция
операция

Монитор

- Взаимное исключение
- Условия блокировки
  - condition variables
    - wait / signal

Три беды

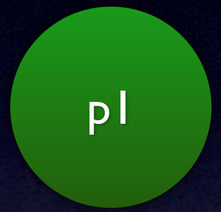
- Starvation
- Deadlock
- Livelock

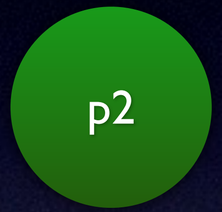
Условия возникновения дедлоков

1. Взаимное исключение
2. hold and wait (процесс блокирует ресурс и ждет освобождения другого ресурса)
3. Система не может “отбирать” ресурсы у процесса
4. Циклическое ожидание (p2 ждет p2, p2 ждет p3, p3 ждет p1)

все условия должны быть соблюдены

пример






```
lock A
…
lock B
…
…
…
unlock B
unlock A
```

```
lock B
…
lock A
…
…
…
unlock A
unlock B
```

Что делать с deadlock?

- Разрешение проблем
- Превентивная работа
- Запретить hold-and-wait *(или пересмотреть блокировки)*
- Запрет циклической блокировки *(уровни доступа к блокировкам)*